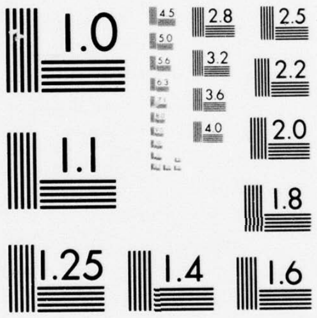
AD-A040 560 YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE F/G 9/2
SKIMMING NEWSPAPER STORIES BY COMPUTER. (U)
MAY 77 G F DEJONG N00014-75-C-1111
UNCLASSIFIED RR-104 NL

1 OF 1
AD A040560

END
DATE
FILMED
7-77


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

12

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

The research described here was done at the Yale Artificial Intelligence Project and is funded in part by the Advanced Research Projects Agency of the Department of Defense and monitored under the Office of Naval Research under contract N00014-75-C-1111.

SKIMMING NEWSPAPER STORIES BY COMPUTER

May 1977

Research Report #104

Gerald F. DeJong

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER #104	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Skimming Newspaper Stories By Computer.		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Gerald F./DeJong		8. CONTRACT OR GRANT NUMBER(s) (15) NØ0014-75-C-1111
9. PERFORMING ORGANIZATION NAME AND ADDRESS Yale University Department of Computer Science 10 Hillhouse Ave., New Haven, Conn. 06520		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		(11) 12. REPORT DATE May, 1977
		13. NUMBER OF PAGES 31
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Program Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited. (9) Research rept.,

(12) 35p. (14) RR-104

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

automated understanding
computational text analysis
knowledge structure
parsing
script
skimming
computational linguistics
conceptual dependency

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes a program that takes a novel approach to understanding natural language. Its domain is newspaper articles and it is unique in that it skims the stories rather than reading them carefully. The program is able to understand the important facts in an article while ignoring the less important text.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

407051

-- OFFICIAL DISTRIBUTION LIST --

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 copies
Office of Naval Research Code 102IP Arlington, Virginia 22217	6 copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois	1 copy
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, California 91106	1 copy
New York Area Office 715 Broadway - 5th Floor New York, New York 10003	1 copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D. C. 20375	6 copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D. C. 20380	1 copy

Office of Naval Research Code 455 Arlington, Virginia 22217	1 copy
Office of Naval Research Code 458 Arlington, Virginia 22217	1 copy
Naval Electronics Laboratory Center Advanced Software Technology Division Code 5200 San Diego, California 92152	1 copy
Mr. E. H. Gleissner Naval Ship Research & Development Center Computation and Mathematics Department Bethesda, Maryland	1 copy
Captain Grace M. Hopper MAICOM/MIS Planning Board (OP-916D) Office of Chief of Naval Operations Washington, D. C. 20350	1 copy
Mr. Kin B. Thompson Technical Director Information Systems Division (OP-91T) Office of Chief of Naval Operations Washington, D. C. 20350	1 copy
Advanced Research Projects Agency Information Processing Techniques 1400 Wilson Boulevard Arlington, Virginia 22209	1 copy
Professor Omar Wing Columbia University in the City of New York Department of Electrical Engineering & Computer Science New York, New York 10027	1 copy

SKIMMING NEWSPAPER STORIES BY COMPUTER

Gerald DeJong
Computer Science Department
Yale University
New Haven, Connecticut 06520

ABSTRACT

This paper discribes a program that takes a novel approach to understanding natural language. Its domain is newspaper articles and it is unique in that it skims the stories rather than reading them carefully. The program is able to understand the important facts in an article while ignoring the less important text. Understanding is done with a construct called a sketchy script. Both the parser and sketchy scripts are discribed in some detail. After understanding an article, the program can produce a brief summary in English, Russian, or Spanish.

The research described in this paper was supported by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under contract N00014-75-C-1111.

SKIMMING NEWSPAPER STORIES BY COMPUTER

Gerald DeJong

INTRODUCTION

FRUMP (Fast Reading Understanding and Memory Program) is a program being developed at Yale to understand newspaper stories. FRUMP was designed to overcome some of the problems that arose from the SAM (Script Applier Mechanism) system over the last several years. Like SAM, FRUMP is a script based understander. The domain of both programs is newspaper stories. However, they are very different. FRUMP is a newspaper skimming program rather than a program that carefully reads text. FRUMP's goal is to understand articles at a rate and depth similar to a human skimming the text. FRUMP can understand and produce a brief summary of a 150 word news articles taken directly from a newspaper in about 5 seconds of CPU time on a DEC KA10 processor.

FRUMP's job is to read newspaper stories, decide whether the stories are new or updates of news events that it has already seen, and store the important information from the article. FRUMP can then give information on a news event by means of different length summaries. So far, on a limited number of domains the results produced by FRUMP are very promising. If progress continues at the current pace, FRUMP will evolve into something of a rarity in the AI world - a working practical program.

SCRIPTS

Recently there have been two realizations that have resulted in major theoretical advances in automated natural language understanding. The first is that vast amounts of world knowledge are required for understanding and the second is that understanding connected text is fundamentally different from understanding isolated sentences [Schank 1974]. Together these indicate that problem solving type structures are necessary for understanding connected natural language text. One such structure developed by Roger Schank and Robert Ableson at Yale is the script [Schank and Abelson 1977]. Scripts are data structures used to represent world knowledge about situations and events where there is very little high level problem solving neccessary. That is, events that are very stylized; where the event happens nearly the same each time or at least where the number of different ways it can happen are small. The domain of both SAM and FRUMP was chosen to be newspaper stories because they often describe highly structured events. Scripts can be used only for events that occur in a very stylized fashion. Fires, plane crashes, earthquakes, mid-east fighting, deaths of government heads, train wrecks etc. all fit into scripts rather well. SAM was the first program to implement that idea, and was conceived to test how well connected text could be understood with scripts. SAM's goal is complete understanding of the text. This is a reasonable test of scripts as understanding structures since people certanly can understand all of a newspaper article. However, people can also skim articles for their high points. They can ignore more or less depending on how much time they wish to spend reading. SAM was not built with this kind of understanding ability.

SAM is also very slow. It sometimes takes as much as fifteen minutes to understand an article of only several paragraphs. This is an obvious practical problem. It is also a theoretical problem: people can easily keep up with daily newspapers by skimming text for a partial understanding; there is no obvious way to retract SAM from complete understanding to partial understanding in order to speed it up to a practical level.

SAMPLE RUN OF FRUMP

The following shows FRUMP understanding a vehicle accident story and then processing an update article. The update adds one important fact (the person responsible) and provides updated information about another (the death toll). These are two of the three types of updates that are discussed in the section on updating. In this example it took 7.5 seconds of CPU time to understand the original story and 10.8 for the update. Comments are in lower case.

```
INPUT:
        10 - 11 CHIHUAHUA, MEXICO, -- A PASSENGER TRAIN CARRYING
TOURISTS, INCLUDING SOME AMERICANS, COLLIDED WITH A FREIGHT
TRAIN IN THE RUGGED SIERRA MADRE OF NORTHERN MEXICO, KILLING AT
LEAST SEVENTEEN PERSONS AND INJURING 45, THE POLICE REPORTED
TODAY.
        THEY SAID THAT AT LEAST FIVE OF THE INJURED WERE
AMERICANS, AND THERE WERE UNOFFICIAL REPORTS THAT ONE OF THE
DEAD WAS FROM NEW YORK CITY.
        SOME OF THE PASSENGERS WERE TRAVEL AGENTS, MOST FROM
MEXICO CITY, MAKING THE TRIP AS PART OF A TOURISM PROMOTION,
THE POLICE SAID.
        THE AMERICAN SOCIETY OF TRAVEL AGENTS HAD BEEN MEETING IN
GUADALAJARA, THOUGH IT WAS NOT KNOWN WHETHER ANY OF THE GROUP
WERE ABOARD THE TRAIN.
        ONE OBSERVATION CAR ON THE RAILROAD TO THE PACIFIC
TUMBLED INTO A 45 FOOT CANYON WHEN THE PASSENGER TRAIN SMASHED
INTO THE FREIGHT YESTERDAY AFTERNOON NEAR THE VILLAGE OF
PITTORREAL ABOUT 20 MILES WEST OF CHIHUAHUA CITY AND 200 MILES
SOUTH OF THE UNITED STATES BORDER, THE POLICE SAID.
        THEY SAID THAT RESCUE WORKERS WERE STILL TRYING TO PRY
APART THE THE CAR'S WRECKAGE TO REACH PASSENGERS TRAPPED INSIDE.
THE RESCUE SQUADS COULD NOT USE CUTTING TORCHES ON THE WRECKAGE
BECAUSE SPILLED DIESEL FUEL MIGHT IGNITE, THE POLICE REPORTED.

SELECTED SKETCHY SCRIPT $VEHACCIDENT

DONE PROCESSING
SATISFIED REQUESTS:

((<=> ($DATELINE LOC &DLOC MONTH &MON DAY &DAY)))
&DLOC
        CLASS       (#LOCATION)
        LOCALE      (*MEXICO*)
        SATISFIED           ((10) (11))
&MON
        NUMBER      (10)
        SATISFIED           (NIL (11))
        CLASS       (#NUMBER)
&DAY
        NUMBER      (11)
        SATISFIED           (NIL NIL)
        CLASS       (#NUMBER)

((<=> ($VEHACCIDENT VEH &VEH OBJ &OBJ LOC &LOC)))
&VEH
        CLASS       (#PHYSOBJ)
        TYPE        (*VEHICLE*)
        SROLE       (&TRAIN)
        SCRIPT      ($TRAIN)
        SATISFIED           ((10) (11))
&OBJ
        CLASS       (#PHYSOBJ)
```

```
          TYPE       (*VEHICLE*)
          SROLE      (&TRAIN)
          SCRIPT     ($TRAIN)
          SATISFIED           ((10) (11))
&LOC
          CLASS      (#LOCATION)
          LOCALE     (*MEXICO*)
          SATISFIED           ((10) (11))

((ACTOR &DEADGRP TOWARD (*HEALTH* VAL (-10))))
&DEADGRP
          NUMBER     (17)
          SATISFIED           ((10) (11))
          CLASS      (#PERSON)

((ACTOR &HURTGRP TOWARD (*HEALTH* VAL (-LT10))))
&HURTGRP
          NUMBER     (45)
          SATISFIED           ((10) (11))
          CLASS      (#PERSON)


CPU TIME = 7.522 SECONDS

SUMMARY:
17 PEOPLE WERE KILLED AND 45 WERE INJURED WHEN A TRAIN CRASHED
INTO A TRAIN IN MEXICO.
INJURED.




INPUT:
          10 - 12 CHIHUAHUA, MEXICO, -- OFFICIAL HERE SAID TODAY
THAT THE DEATH TOLL FROM THE COLLISION OF AN EXCURSION TRAIN
WITH A FREIGHT TRAIN ON SUNDAY HAS RISEN TO 23.  THE DEAD
INCLUDE TWO AMERICANS.  RESPONSIBILITY FOR THE ACCIDENT WAS
LAID TO THE PASSENGER TRAIN ENGINEER WHO APPARENTLY FAILED TO
HEED A STOP SIGNAL.
          THE DISTRICT ATTORNEY'S OFFICE SAID TWO OTHER AMERICANS
HAD BEEN INJURED IN THE COLLISION NEAR THE BARRANCA DEL COBRE
IN THE SIERRA MADRE.
          ALL THE DEAD WERE MEXICANS EXCEPT THE TWO AMERICANS AND
TWO BRITONS.  MOST OF THOSE ABOARD WERE MEXICAN TRAVEL AGENTS.

          THE AMERICANS WERE IDENTIFIED AS MART MORTELLARO OF NEW
YORK AND MARTIN WARD WHOSE HOMETOWN WAS STILL NOT AVAILABLE.
THE DISTRICT ATTORNEY'S OFFICE SAID THE TWO INJURED AMERICANS
WERE PAUL JOSEPH CALLSEN AND MARY CALLSEN, BOTH OF NEW YORK.

SELECTED SKETCHY SCRIPT $VEHACCIDENT
UPDATE OF PREVIOUS NEWS EVENT

DONE PROCESSING
```

```
SATISFIED REQUESTS:

((<=> ($DATELINE LOC &DLOC MONTH &MON DAY &DAY)))
&DLOC
        SATISFIED          ((10) (12))
        LOCALE     (*MEXICO*)
        CLASS      (#LOCATION)
&MON
        CLASS      (#NUMBER)
        SATISFIED          ((10) (12))
        NUMBER     (10)
&DAY
        CLASS      (#NUMBER)
        SATISFIED          ((10) (11))
        NUMBER     (12)

((<=> ($VEHACCIDENT VEH &VEH OBJ &OBJ LOC &LOC)))
&VEH
        CLASS      (#PHYSOBJ)
        TYPE       (*VEHICLE*)
        SROLE      (&TRAIN)
        SCRIPT     ($TRAIN)
        SATISFIED          ((10) (12))
&OBJ
        CLASS      (#PHYSOBJ)
        TYPE       (*VEHICLE*)
        SROLE      (&TRAIN)
        SCRIPT     ($TRAIN)
        SATISFIED          ((10) (12))
&LOC
        CLASS      (#LOCATION)
        LOCALE     (*MEXICO*)
        SATISFIED          ((10) (12))

((ACTOR &DEADGRP TOWARD (*HEALTH* VAL (-10))))
&DEADGRP
        CLASS      (#PERSON)                    the number killed has been

        SATISFIED          ((10) (12))          modified to 23   this is a

        NUMBER     (23)                         type two update as described
                                                in the section on updating
((ACTOR &HURTGRP TOWARD (*HEALTH* VAL (-LT10))))
&HURTGRP
        CLASS      (#PERSON)
        SATISFIED          ((10) (11))
        NUMBER     (45)

((<=> ($FAULT ACTOR &FAULT)))
&FAULT
        CLASS      (#PERSON)                    this request was not present
        SROLE      (&ENGINEER)                  in the previous print out
        SCRIPT     ($TRAIN)                     because it was not satisfied
        SATISFIED          ((10) (12))          by the first article   this
                                                is a type one update

CPU TIME = 10.798 SECONDS
```

SUMMARY:
23 PEOPLE WERE KILLED AND 45 WERE INJURED WHEN A TRAIN CRASHED INTO A TRAIN IN MEXICO. THE ENGINEER WAS RESPONSIBLE FOR THE ACCIDENT.

OVERVIEW OF HOW FRUMP UNDERSTANDS

FRUMP was developed to provide fast partial understanding. FRUMP's job is to skim newspaper articles with a clear idea of what and how much it wants to get out of them. To achieve this, instead of using a script like SAM's, FRUMP uses what we call a sketchy script. The crucial difference is that sketchy scripts have far fewer conceptual dependancy representations (only those corresponding to the most important events in SAMs scripts) and more often than not, the causal connections between conceptualizations are not included. The result is that FRUMP understands most of what is important to understand in news articles and works very much faster than SAM. The article of several paragraphs that takes SAM a quarter of an hour to understand can be processed by FRUMP in under ten CPU seconds. FRUMP is written in Irvine Lisp and runs interpretively on a Digital Equipment Corporation KA10 processor.

When FRUMP begins to read a newspaper story, it already knows what facts it wants to find. For each type of newspaper story, FRUMP has a list of expected facts that it wants to see. These expectations are called "requests". The collection of all the requests for one type of story makes up the "sketchy script" for that story type. In the remainder of the paper when I refer to a script I will mean FRUMP's sketchy scripts not SAM's scripts unless otherwise noted.

In understanding an article, FRUMP must select a script and then try to find occurrences in the article of the facts represented by the requests. Requests are in conceptual dependancy format and contain unfilled slots. These slots are called "script variables". Understanding an article consists of finding the information corresponding to a request in the text and filling in the slots (binding the script variables) in that request. When an instance of one of the requests is found in the text and the script variables have been bound, that request is said to be satisfied. The process of satisfying the requests of a script is called instantiating that script. The number of requests in each script is small. The requests correspond to the most important information in a particular type of story. For example, the vehicle accident sketchy script used in the sample run above contains four requests. The first request in the vehicle accident script will be satisfied when FRUMP finds the type of vehicle in the accident, the object that the vehicle collided with, and the location of the accident. FRUMP can satisfy the second request by finding the number of people killed; the third by the number injured, and the fourth by who was at fault in the accident. When all of these requests are satisfied by a story, FRUMP knows all that it wants to know about that news event. The rest of the article will be ignored.

When FRUMP is given a new article to understand it skims the first paragraph for identifying information. This information is used to find the appropriate script to use to understand the article. Once the script is identified, FRUMP begins skimming the article.

FRUMP is composed of two conceptually different parts: a parser and a script applier. The parser FRUMP employs was inspired by Becker's phrasal lexicon [Becker 1975] which was presented at the TINLAP conference in 1975 but unfortunately was not actively pursued by him after that. The parser parses phrases from the text into conceptual dependancy representations. The script applier then matches these conceptual representation against the requests in the script. When a match is found, the fillers in the parser representation are used to bind the script variables occurring in the request.

FRUMP uses the same language free system of representations as is used by Riesbeck's parser [Riesbeck 1974] which is used in SAM. Yet FRUMP's parser is very different from the parser in the SAM system. The Riesbeck parser parses an entire sentence at a time. There is very little communication allowed with the script applier of SAM during parsing: FRUMP's parser is concerned only with parsing parts of a sentence. In SAM the parser and script applier are very distict. As a result, each does its thing with little influence over the other. FRUMP, however, is much more integrated. The parser and script applier do different tasks but the precise division between the two is fuzzy. The advantage of the fuzzyness is that the two modules can communicate with each other freely. This allows the script applier to control parsing to an extent not possible in SAM. FRUMP's script applier can tell the parser which of several interpretations is correct or even to stop trying to parse the current input text.

THE CONCEPTUAL FRAGMENT PARSER

The parser is very much top down; it is driven by the high level requests of the sketchy scripts and works very closely with the script applier. The parsing stradegy is to find conceptual fragments from the input text being skimmed that will satisfy all or part of some script request. Important properties of the actual conceptual referents in the text are then copied to variables in the conceptual representation from FRUMP's dictionary. Finally, this conceptual representation of the meaning of the input text is returned as the parse. Dictionary entries for each word are made up of a list of meanings fragments in conceptual dependency format. For each different meaning there is a series of context tests that must be satisfied before this meaning can be realized as the parse. There are also instructions on how to bind variable role fillers in the conceptual dependency representation. FRUMP has two dictionaries: one is the conceptual fragment dictionary discribed below; the other contains information about objects and forms and tenses of entries in the other dictionary

The following is the conceptual fragment dictionary entry for "strike".

```
( (BKWRDS (M1 (TYPE (*VEHICLE*))))
  (FRWRDS (M2 (CLASS (#PHYSOBJ))))
  ((MERGE P1 M1) (MERGE P2 M2))
  ((<=> ($VEHACCIDENT VEH P1 OBJ P2))) )
```

The context tests are arranged in two lists which search backward and forward respectively from the entry word for words that will satisfy the context tests. If a context test is an atom, that atom must be present in the input text. If the contex test is a list it is satisfied by finding a word in the input text that has all the properties specified, and copying them to a temporary variable. For example, the first line in the above dictionary entry searches backward from a form of the word "strike" for a vehicle. If it finds one, all the properties of that vehicle are copied to the temporary variable M1 and the forward test are evaluated. The forward tests require the word "strike" be followed by a physical object. If all the context tests are satisfied, the properties of the temporary variables are copied to the role fillers in the conceptual representation. This representation is then returned as the parse.

The output of FRUMP's parser is not modified English as it is in Colby's system [Parkison, Colby, and Faught 1976], but a language free conceptual representation. The advantage of a language free representation is that different phrases with the same meaning will be parsed into the same representation. This in turn means that the test to see if a request is satisfyed by a parse is very efficient. It also makes generating summaries in different languages no harder than generating the summary in English.

EXAMPLE OF A SKETCHY SCRIPT

One of the scripts that FRUMP has is a vehicle accident script. All vehicle accidents whether they are train wrecks, boat collisions, plane crashes etc. have many things in common. In a vehicle accident story there is always a vehicle and there is always an object that it collides with. There is always the possibility that a number of people are killed or injured. In addition, in newspaper stories, the cause of the collision is often reported. These are the important points of a vehicle accident and these are what FRUMP tries to find out when reading a vehicle accident article. There are, of course, many other things that can happen in a collision. For example, the injured people are often taken to a hospital, for auto crashes there is often a policeman called to the scene etc. These are less important facts and unless there is some special reason for noticing them, a human skimming the article will usually miss them. FRUMP also ignores these lesser points. The vehicle accident script currently consists of four requests at three importance levels.

```
Request R1
        value:   ((((<=> ($VEHACCIDENT VEH &VEH OBJ &OBJ LOC &LOC)))
                   ((EQU (<=>) $VEHACCIDENT))
                   (PROP (<=> VEH) *VEHICLE* TYPE)
                   (PROP (<=> OBJ) #PHYSOBJ CLASS)
                   (PROP (<=> LOC) #LOCATION CLASS))
        importance: 0

Request R2
        value:   (((ACTOR &DEADGRP TOWARD (*HEALTH* VAL (-10))))
                   ((EQU (TOWARD) *HEALTH*)
                    (EQU (TOWARD VAL) -10))
                   (PROP (ACTOR) #PERSON CLASS))
        importance: 1

Request R3
        value:   (((ACTOR &HURTGRP TOWARD (*HEALTH* VAL (-LT10))))
                   ((EQU (TOWARD) *HEALTH*)
                    (EQU (TOWARD VAL) -LT10))
                   (PROP (ACTOR) #PERSON CLASS))
        importance: 1

Request R4
        value:   (((<=> ($FAULT ACTOR &ACTOR)))
                   ((EQU (<=>) $FAULT))
                   (PROP (<=> ACTOR) #PERSON CLASS))
        importance: 2
```

Before FRUMP's understanding can be understood in detail, one must understand the requests of its sketchy scripts. As a typical example of FRUMP's requests consider request R2 above. The first line of the request is the conceptual dependancy representation of the request. &DEADGRP is one of variables of the vehicle accident script. (*HEALTH* VAL (-10)) is the conceptual dependancy representation for dead. When the variable &DEADGRP is bound to something, the meaning of this conceptual dependancy representation is that the something it gets bound to is dead. The next three lines are constraint tests that are applied to the output of FRUMP's parser. If the parser yields a representation that passes all of these tests, the script variables contained in the request are bound to the corresponding conceptual role fillers in the parser representation and the request is satisfied. The first test in the example request checks that the filler of the TOWARD role in the parser representation is *HEALTH*. The second test checks that the filler of the VAL slot in the filler of the TOWARD slot is -10. The third test checks that the filler of the ACTOR slot (that is the thing that the parser proposes should be bound to &DEADGRP in the request) is of conceptual type PERSON. This means that the only thing that can be bound to &DEADGRP is a person or group of people. In fact the parser is set up so that only groups will be generated in this slot. One of the important properties of groups is the number of things in them. When &DEADGRP is bound to something, all its properties are copied over to the script variable &DEADGRP. Therefore one of the pieces of information available from this request after it is satisfied (and indeed the most important datum of this request) is the number of people who were killed. The observant reader will have realized that the first two tests in the request are set off from the third one by parentheses. This is to group the tests by whether or not they involve one of the script variables. The first two tests in the example do; the third does not. The grouping makes the understanding process more efficient. This will become clear in the section HOW FRUMP UNDERSTANDS.

The requests in scripts are actually global variables in LISP. The structures like the example above are the values of these global variables. This particular request appears in several scripts. It is request R2 in the vehicle accident script and request R4 in the earthquake script. The global variables are marked with information about their requests. When a request is satisfied, a property is put on the global variable marking it as satisfied. The global is also tagged with the date of the story being read that satisfied it. This is done for updating purposes. One type of updating entails replacing some of the role fillers in a previously satisfied request. When this is done, FRUMP must be sure that the new role fillers are more current than the old ones.

FRUMP can understand an article at several levels. This is done by tagging each individual request with an importance level. The global variable is tagged and not the representation, of course, so the same conceptual representation for a request can appear in several scripts and have a different importance value in each. Importance levels are numbers; smaller numbers denote more important requests. When FRUMP is started, an importance level can be associated with each script. When FRUMP decides on a script to use for understanding a particular story, the importance level at which to understand that story is then also set. Each script corresponds to a different story type so this amounts to telling FRUMP at what level to understand different story types. Each request below the set importance level is ignored by FRUMP. Since the ignored requests are not processed, FRUMP can run slower or faster finding out more or less information about a news event.

HOW FRUMP UNDERSTANDS

Once FRUMP has the correct script to use, it starts to scan the article looking for conceptual fragment words. When it finds one, it retrieves the dictionary entry for that word. Recall that the dictionary entry consists of a list each element of which contains context tests and a representation that corresponds to one meaning. FRUMP tries to realize each meaning one at a time until all the context tests are satisfied or the dictionary list is exhausted.

The processing of each dictionary word sense or possible meaning consists of first making a list of all the outstanding requests that the conceptual representation of this meaning might satisfy. This is done to avoid evaluating all of the context tests of a word sense that has no chance of satisfying a request, and to limit the number of requests that the parse need be matched against. A request is included in the list only if all of the first group of role filler tests of this request are satisfied. Remember that the first group of role filler tests are all the tests that do not reference one of the script variables. Therefore these tests can be made before the script variables or the variables in the context tests are bound. For example, if while processing a vehicle accident story, the parser found a word that indicated ((ACTOR P1 TOWARD (*HEALTH* VAL (-10)))) might be a parse, the list of possible requests would be just (R2). The tests that do not look at script variables require that the <=> filler for R1 be $VEHACCIDENT, the filler of the VAL slot in the TOWARD slot must be -LT10 for R3, and the <=> slot in R4 must be filled with $FAULT. Thus the only request that might be satisfied is R2. If the list is empty, there is no need to evaluate any of the context tests; it cannot satisfy a high level request. At this point, none of the context tests have been evaluated so that this word sense might not be correct. However, the list is very cheap to create and usually cuts down on processing later. This is the reason for separating the request tests that reference script variables from those that do not. If the list contains at least one element, the context tests for this meaning are evaluated one at a time. If there is at least one context test that fails, this sense is not a proper reading of the text. If all the word senses fail, the word that was found was a false alarm, and FRUMP reverts to the original scan. If all the context tests are satisfied, the conceptual representation is filled out with the variables bound while evaluating the context tests. This representation is then matched against elements of the list of potentially satisfiable requests made earlier. If one matches, the script variables in that request are bound to the role fillers of the representation, the request is marked with the date of the article, and it is marked as satisfied. At this point, FRUMP can dynamically modify the sketchy script. Associated with each script variable can be a set of demons which are checked when the script variable is satisfied. They can make arbitrary tests and load or delete requests. Thus it is possible, for example, to have FRUMP look for aid to a

GY4Q;P6 V2 V; V1 XV; U6 J 1KOKPK KJP;XO4J5K U4; QY; V2 V; 8J1 J TVDL O4J5K" <XV1 JTY4Q;1 ;Y XVNX DKOKD VQ2KPKQGVQN JQL TJ5K1 >'^%} much more efficient by eliminating the need to process large numbers of very specialized requests; the specialized requests are not loaded until they are needed. After processing the satisfied request, FRUMP continues its scan for conceptual fragment words. If no request fulfills the detailed match, the FRUMP reverts to the original text scan. Notice that there were three ways parsing can be discontinued; in these cases FRUMP does just enough work to realize it is working on a bad parse. This is in a large way responsible for FRUMP's efficiency in processing and is directly attributable to the broad communication between the parser and the control structure that makes up the script applier.

When FRUMP is finished processing an article, some requests will have been satisfied but, very likely, others will not. The script has been partially instantiated. This partially instantiated script is then stored on a disk file. If FRUMP should later come across an article updating this news event, it can then retrieve this partially instantiated script and continue satisfying requests where it left off.

Presented with an article, FRUMP chooses one of the three following ways to process it. First, it can decide that the article is an update of a news event that it has previously processed and select the partially instantiated script from that article to understand with. Second, it can decide that it is the first article of a news event and select the appropriate virgin script. Third, it can fail to recognize the article as one of the types of events for which there exists a script in which case it will ignore the entire article. The choice is made from information gleaned from a preliminary scan of the article's first paragraph. This scan is made with a special set of active requests.

There is for each script one key request which if satisfied in the text, strongly indicates that its script is appropriate to understand the article. For example, the key request for the vehicle accident script is ((<=> ($vehaccident veh &veh obj &obj loc &loc))) here &veh, &obj, and &loc are script variables which get bound to the vehicle, the object collided with, and the location of the accident respectively. Furthermore, due to the style of newspaper writers, this request seems always to be satisfied in the first paragraph (and usually the first sentence). The special set of requests is therefore composed of the key request from each virgin script that FRUMP has.

The first paragraph is skimmed until one key request is satisfied. FRUMP now knows which script type the story is and also some information about the story. In the case of the vehicle accident it knows what the vehicle and object are and the location where the accident occurred. This information is used to decide if the current article refers to a previous news event or a new one. After a sketchy script is partially instantiated by the first article of an event, it is stored away. The type of script it is and the key information about the event are stored specially. After a new article's first paragraph is skimmed, the key information gained is matched against all stored scripts of the same type. If a stored partially instantiated script is found that matches, it is brought into core and used to understand the new article. If no previous script is matched, a virgin script with no requests satisfied is used to understand the story. When it is finished, FRUMP writes this partially instantiated script out on the disk file so that any update articles that it finds will have access to it. UPDATING SCRIPTS

There are three main types of updates that FRUMP must handle. These types correspond to pieces of information and not to articles so that an update article can cause more than one type of update to be made. The update types differ from one another by how the new information is added to the partially instantiated sketchy script.

In the first kind of update, information is only added to a sketchy script. That is, a new article is found to refer to the same news event as previous articles and it supplies information that satisfies a request that was never before satisfied. This is the simplest type of update and is handled as follows: After FRUMP finishes an article, the sketchy script is written out to a file with the key identifying information discussed above. Some of the requests will have been satisfied and some will not. All the requests whether satisfied or not are written on the file. When this partially instantiated script is read back into core, the unsatisfied requests are, of course, still active. On reading the update article then, this type of update is treated exactly as if it were a virgin script. When a previously unsatisfied request is satisfied, it is marked as satisfied and tagged with the date of the newspaper. An example of this type of update can be found in the sample run of FRUMP. The request looking for the person who caused the accident was left unsatisfied by the first article. When the second article was skimmed, this request was matched and satisfied.

In the second type of update, the information in the update article replaces the information in an already satisfied request. In this type, generally only one request is changed at a time and the change is a direct modification of one or more role fillers of the request. All requests whether satisfied or not are processed during understanding as if they are not. When the role fillers of a request are to be bound, the date that they were last bound is compared to the date of the current article. If the current article is later, the fillers are updated. If not, the information from the current article is thrown away. An example of this kind of update is the revision of the death toll by the update article in the sample FRUMP run.

The third type of update is the most complicated. There are many news event types where an arbitrary number of similar sub-events can occur. These sub-events themselves may be rather complex. For example, an earthquake may be followed by any number of aftershocks. Each aftershock may cause death and injury. The recent fighting in Lebanon was made up of a number of individual clashes. Oil from a leaking tanker can wash ashore in several places at different times; each causing different kinds and varying degrees of damage to the shoreline. There are three things to notice about such updates. First, they add new rather than replacing old information. Therefore, they must be processed by as yet unsatisfied requests. Second, the structure of each sub-event can be complex enough so that it can not be represented by one request alone. Third, since the initial requests for each sub-event must be the same, there is the possibility for any number of copies of the same request to exist in a script each satisfied by a different sub-event. For example, an earthquake and two of its aftershocks may all cause people to be killed. In this earthquake script then, there will be three copies of the request ((ACTOR &DEADGRP <=> (*HEALTH* VAL (-10)))) each satisfied with a different &DEADGRP.

The solution to these problems of the third update type is to organize the requests corresponding to sub-events into bundles. The script will always have one fresh copy of the bundle active and completely uninstantiated. When a bundle is about to be partially instantiated (that is when at least one of its requests is to be satisfied) a copy of the uninstantiated bundle is made and these new requests are added to the script. Then FRUMP continues instantiating the bundle. This will enable FRUMP to understand any number of the sub-events. Of course, an article could update an update article (e.g. revising the death toll caused by an aftershock of an earthquake). If one of the requests in a particular bundle has to be changed, FRUMP must first identify the bundle. After that the update can be handled exactly as the type two updates above.

Bundles are subsections or scenes of scripts. They are very similar to scripts in many ways. In particular, they can always be differentiated by key information. This key information is often simply the date or location of the sub-event. For example, a newspaper report might update the death toll from last Thursday's aftershock of the earthquake that struck Eastern Turkey five days ago. Five days ago, Eastern Turkey, and the fact that it is an earthquake are used to find the original script. Within this script, FRUMP then finds the bundle of requests for the proper aftershock by matching the date of each to the date last Thursday. When it finds the correct bundle, FRUMP finds the request corresponding to the number of people killed and updates the proper script variable.

MORE FRUMP EXAMPLES

The following news stories were taken directly from the New York Times and the New Haven Register. The Spanish summarizer was written by Jaime Carbonell and the Russian summarizer by Anatole Gershman.

```
INPUT:
     2 - 4 ITALY -- A SEVERE EARTHQUAKE STRUCK NORTHEASTERN
ITALY LAST NIGHT, COLLAPSING ENTIRE SECTIONS OF TOWNS NORTHEAST
OF VENICE NEAR THE YUGOSLAV BORDER, KILLING AT LEAST 95 PERSONS
AND INJURING AT LEAST 1000, THE ITALIAN INTERIOR MINISTRY
REPORTED.
     IN THE CITY OF UDINE ALONE, A GOVERNMENT SPOKESMAN SAID
THEY FEARED AT LEAST 200 DEAD UNDER THE DEBRIS.  THE CITY, ON
THE MAIN RAILROAD BETWEEN ROME AND VIENNA, HAS A POPULATION OF
ABOUT 90000.
     THE SPOKESMAN FOR THE CARIBINIERI, THE PARAMILITARY
NATIONAL POLICE FORCE, SAID THERE HAD BEEN REPORTS OF SEVERE
DAMAGE FROM HALF A DOZEN TOWNS IN THE FOOTHILLS OF THE ALPS,
WITH WHOLE FAMILIES BURIED IN BUILDING COLLAPSES.
COMMUNICATIONS WITH A NUMBER OF POINTS IN THE AREA WERE STILL
OUT.
     THE EARTHQUAKE WAS RECORDED AT 6.3 ON THE RICHTER SCALE,
WHICH MEASURES GROUND MOTION.  IN POPULATED AREAS, A QUAKE
REGISTERING 4 ON THAT SCALE CAN CAUSE MODERATE DAMAGE, A
READING OF 6 CAN BE SEVERE AND A READING OF 7 INDICATES A MAJOR
EARTHQUAKE.

SELECTED SKETCHY SCRIPT $EARTHQUAKE

DONE PROCESSING
SATISFIED REQUESTS:

((<=> ($DATELINE LOC &DLOC MONTH &MON DAY &DAY)))
&DLOC
     CLASS      (#LOCATION)
     LOCALE     (*ITALY*)
     SATISFIED          ((2) (4))
&MON
     NUMBER     (2)
     SATISFIED          (NIL (4))
     CLASS      (#NUMBER)
&DAY
```

```
      NUMBER     (4)
      SATISFIED          (NIL NIL)
      CLASS      (#NUMBER)

((<=> ($EARTHQUAKE LOC &LOC SEVERITY &RIC)))
&LOC
      CLASS      (#LOCATION)
      LOCALE     (*ITALY*)
      SATISFIED          ((2) (4))
&RIC
      NUMBER     (6.3)
      SATISFIED          ((2) (4))
      CLASS      (#NUMBER)

((ACTOR &DEADGRP TOWARD (*HEALTH* VAL (-10))))
&DEADGRP
      NUMBER     (95)
      SATISFIED          ((2) (4))
      CLASS      (#PERSON)

((ACTOR &HURTGRP TOWARD (*HEALTH* VAL (-LT10))))
&HURTGRP
      NUMBER     (1000)
      SATISFIED          ((2) (4))
      CLASS      (#PERSON)


CPU TIME = 9.440 SECONDS


RUSSIAN SUMMARY:
ZEMLETRYASENIE SREDNEI SILY PROIZOSHLO V ITALII.  CILA
ZEMLETRYASENIYA OPREDELENA V 6.3 BALLA PO SHKALE RIKHTERA.  PRI
ZEMLETRYASENII 95 CHELOVEK BYLO UBITO I 1000 RANENO.

SPANISH SUMMARY:
HUBO 95 MUERTOS Y 1000 HERIDOS EN UN TERREMOTO FUERTE EN ITALIA.
EL TERREMOTO MIDIO 6.3 EN LA ESCALA RICHTER.

ENGLISH SUMMARY:
95 PEOPLE WERE KILLED AND 1000 WERE INJURED IN A SEVERE
EARTHQUAKE THAT STRUCK ITALY.  THE QUAKE REGISTERED 6.3 ON THE
RICHTER SCALE.



INPUT:
      11 - 29 KATHEKANI, KENYA, -- AT LEAST 12 PEOPLE WERE
REPORTED KILLED EARLY TODAY WHEN AN EXPRESS TRAIN RAN ONTO A
FLOODED BRIDGE WHOSE RAILS HAD BEEN SWEPT AWAY, CRASHED THROUGH
IT AND PLUNGED INTO A RIVER IN KENYA.
      THE OFFICIAL PRESS AGENCY REPORTED THAT THE DEATH TOLL
WAS AT LEAST 12 AND THAT 70 WERE INJURED IN WHAT RAILROAD
OFFICIALS CALLED THE WORST PASSENGER TRAIN DISASTER IN EAST
```

```
AFRICAN HISTORY.

SELECTED SKETCHY SCRIPT $VEHACCIDENT

DONE PROCESSING
SATISFIED REQUESTS:

((<=> ($DATELINE LOC &DLOC MONTH &MON DAY &DAY)))
&DLOC
        CLASS      (#LOCATION)
        LOCALE     (*KENYA*)
        SATISFIED           ((11) (29))
&MON
        NUMBER     (11)
        SATISFIED           (NIL (29))
        CLASS      (#NUMBER)
&DAY
        NUMBER     (29)
        SATISFIED           (NIL NIL)
        CLASS      (#NUMBER)

((<=> ($VEHACCIDENT VEH &VEH OBJ &OBJ LOC &LOC)))
&VEH
        CLASS      (#PHYSOBJ)
        TYPE       (*VEHICLE*)
        SROLE      (&TRAIN)
        SCRIPT     ($TRAIN)
        SATISFIED           ((11) (29))
&OBJ
        CLASS      (#PHYSOBJ)
        CONENT     (*RIVER*)
        CPRPS      (*WATER*)
        SATISFIED           ((11) (29))
&LOC
        CLASS      (#LOCATION)
        LOCALE     (*KENYA*)
        SATISFIED           ((11) (29))

((ACTOR &DEADGRP TOWARD (*HEALTH* VAL (-10))))
&DEADGRP
        NUMBER     (12)
        SATISFIED           ((11) (29))
        CLASS      (#PERSON)

((ACTOR &HURTGRP TOWARD (*HEALTH* VAL (-LT10))))
&HURTGRP
        NUMBER     (70)
        SATISFIED           ((11) (29))
        CLASS      (#PERSON)


CPU TIME = 9.539 SECONDS


RUSSIAN SUMMARY:
```

```
V ZHELEZNODOROZHNOI KATASTROFE V KENII 12 CHELOVEK BYLO UBITO I
70 RANENO.

SPANISH SUMMARY:
HUBO UN ACCIDENTE DE FERROCARRIL EN KENYA QUE RESULTO EN 12
MUERTOS Y 70 HERIDOS.

ENGLISH SUMMARY:
A TRAIN CRASH CLAIMED 12 LIVES AND INJURED 70 IN KENYA.




INPUT:
      3 - 4 PISA, ITALY -- OFFICIALS TODAY SEARCHED FOR THE
BLACK BOX FLIGHT RECORDER ABOARD AN ITALIAN AIR FORCE TRANSPORT
PLANE TO DETERMINE WHY THE CRAFT CRASHED INTO A MOUNTAINSIDE
KILLING 44 PERSONS.
      THEY SAID THE WEATHER WAS CALM AND CLEAR, EXCEPT FOR SOME
GROUND LEVEL FOG, WHEN THE U S MADE HERCULES C130 TRANSPORT
PLANE HIT MT.  SERRA MOMENTS AFTER TAKEOFF THURSDAY.
      THE PILOT, DESCRIBED AS ONE OF THE COUNTRY'S MOST
EXPERIENCED, DID NOT REPORT ANY TROUBLE IN A BRIEF RADIO
CONNVERSATION BEFORE THE CRASH.

SELECTED SKETCHY SCRIPT $VEHACCIDENT

DONE PROCESSING
SATISFIED REQUESTS:

((<=> ($DATELINE LOC &DLOC MONTH &MON DAY &DAY)))
&DLOC
      CLASS       (#LOCATION)
      LOCALE      (*ITALY*)
      SATISFIED          ((3) (4))
&MON
      NUMBER      (3)
      SATISFIED          (NIL (4))
      CLASS       (#NUMBER)
&DAY
      NUMBER      (4)
      SATISFIED          (NIL NIL)
      CLASS       (#NUMBER)

((<=> ($VEHACCIDENT VEH &VEH OBJ &OBJ LOC &LOC)))
&VEH
      CLASS       (#PHYSOBJ)
      TYPE        (*VEHICLE*)
      SROLE       (&AIRPLANE)
      SCRIPT      ($AIRPLANE)
      SATISFIED          ((3) (4))
&OBJ
      CLASS       (#PHYSOBJ)
```

```
          CONENT     (*MOUNTAIN*)
          SATISFIED           ((3) (4))
&LOC
          SATISFIED           ((3) (4))
          CLASS      #LOCATION
          LOCALE     (*ITALY*)

((ACTOR &DEADGRP TOWARD (*HEALTH* VAL (-10))))
&DEADGRP
          NUMBER     (44)
          SATISFIED           ((3) (4))
          CLASS      (#PERSON)


CPU TIME = 6.778 SECONDS


RUSSIAN SUMMARY:
V AVIATSIONNOI KATASTROFE V ITALII 44 CHELOVEK BYLO UBITO.

SPANISH SUMMARY:
HUBO 44 MUERTOS CUANDO UN AVION CHOCO CONTRA UN MONTANA EN
ITALIA.

ENGLISH SUMMARY:
44 PEOPLE WERE KILLED WHEN AN AIRPLANE CRASHED INTO A MOUNTAIN
IN ITALY.
```

```
INPUT:
          4 - 14, TOKYO -- CHINESE OFFICIALS HAVE TURNED THE
NORMALLY STAID ANNUAL MEETING OF THE ECONOMIC COMMISSION FOR
ASIA AND THE FAR EAST INTO A TURBULENT POLITICAL FORUM BY OPENLY
QUARRELING WITH THE RUSSIANS AND BY WALKING OUT ON THE OTHER
DELEGATES SPEECHES.
          THE CHINESE DELEGATION, LED BY AN CHIH-VUAN, WALKED OUT
FOR THE THIRD TIME THIS MORNING WHEN THE LEADER OF THE
SOUTH VIETNAMESE DELEGATION, LE TUAN ANH, ROSE TO SPEAK.  THE
CHINESE CHARGED THAT A VIETCONG REPRESENTATIVE SHOULD ALSO HAVE
BEEN INVITED TO THE ECONOMIC COMMISSION, AN UNITED NATIONS UNIT.

          JUST AFTER THE CONFERENCE OPENED ON TUESDAY MR AN, THE
CHINESE DELEGATION LEADER, ACCUSED THE SOVIET UNION OF
'STEPPING UP ITS EXPANSION IN THIS REGION IN AN ATTEMPT TO
SEIZE HEGEMONY'.  HE CHARGED THAT THE SOVIET UNION INTENDED 'TO
CONTROL AND DIVIDE ASIAN COUNTRIES AND INCORPORATE THEM
GRADUALLY INTO ITS SPHERE OF INFLUENCE'.

SELECTED SKETCHY SCRIPT SMEETING
```

```
DONE PROCESSING
SATISFIED REQUESTS:

(((<=> ($DATELINE LOC &DLOC MONTH &MON DAY &DAY)))
&DLOC
        CLASS       (#CITY #LOCATION)
        LOCALE      (*TOKYO*)
        SATISFIED            ((4) (14))
&MON
        NUMBER      (4)
        SATISFIED            (NIL (14))
        CLASS       (#NUMBER)
&DAY
        NUMBER      (14)
        SATISFIED            (NIL NIL)
        CLASS       (#NUMBER)
(((<=> ($MEETING MTGRP &MTGRP)))
&MTGRP
        CLASS       (#ORGANIZATION)
        ORGANIZATION         (*UN*)
        SATISFIED            ((4) (14))

(((<=> ($WALKOUT WKR &WKR WKE &WKE)))
&WKR
        CLASS       (#COUNTRY #LOCATION)
        LOCALE      (*CHINA*)
        SATISFIED            ((4) (14))
&WKE
        CLASS       (#COUNTRY #LOCATION)
        LOCALE      (*SOUTH-VIETNAM*)
        SATISFIED            ((4) (14))

CPU TIME = 3.795 SECONDS

ENGLISH SUMMARY:
IN AN UNTITED NATIONS MEETING, CHINA PROTESTED BY WALKING OUT
ON SOUTH VIETNAM.
```

FUTURE WORK

Future work on FRUMP is divided into short range and long range plans. Short range plans include hooking FRUMP up to one of the major news wire services and adding a a dictionary of superset subset relations. We will also add a large number of new sketchy scripts and plan a major revision of FRUMP's parser. FRUMP will soon have access to the United Press International news wire. This will be very helpful in supplying FRUMP with lots of consistent real world data, and sould be of great help in testing FRUMP's reliability and in choosing new script types to incorporate into the system. The superset subset relations will be useful primarily for providing an efficient way of more fully specifying script variables. For example, FRUMP presently does not know the difference between frieght trains and passenger trains. A simple network of superset subset relations could easily provide this capability. With such a network, FRUMP could activate different requests in a script depending on the type of train involved in the crash.

Long term plans include writing a question answering module for FRUMP and providing the system support for making FRUMP available to any Yale user. A user would then be able to peruse standard length summaries of wire service articles, ask for longer summaries or perhaps the original of a particular article, or ask questions about a specific article or news subject. In addition, FRUMP could maintain a list of subjects that individual users are interested in. Then when an article of interest that would update a users current knowledge comes over the wire, FRUMP could notify that user.

The short range goals are currently being worked on and ought to be incorporated into FRUMP in a few months. The long range goals might take as long as a year or two. However, barring unforeseen difficulties FRUMP has a fine chance of becoming a working practical application of natural language research.

REFERENCES

Becker, J.,"The Phrasal Lexicon" Proc. Theoretical Issues In Natural Language Processing Workshop, Cambridge, Mass. June, 1975.

Parkison, R., Colby, M., and Faught, W., "Conversational Language Comprehension Using Inegrated Pattern-Matching and Limited Parsing" Technical Report, UCLA Psyciatry Department, Los Angles, Ca., 1976.

Reisbeck, C. K.,"Computational Understanding: Analysis of Sentences and Context" Ph.D. thesis, AI Memo AIM-238, Stanford University, Palo Alto, Ca., 1974.

Schank, R. C.,"Understanding Paragraphs" Technical Report, Instituto per gli studi Semantici e Cognitivi, Castagnola, Switzerland, 1974.

Schank, R. C. and Abelson, R. P.,Scripts, Plans, Goals, and Understanding Hillsdale, New Jersey: Lawrence Earlbaum Press, 1977.